REMARKS

This amendment is in response to the examiner's office action dated 10/27/2003 and further in view of the interview of 02/13/2004. Applicants are appreciative of the professional and courteous interview held with the examiner. Applicant believes that the arguments presented in the interview and the current amendment should obviate outstanding issues and make the remaining claims allowable. Reconsideration of this application is respectfully requested in view of the foregoing amendment and the remarks that follow.

STATUS OF CLAIMS

Claims 1-68 are pending.

Claims 1-68 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Norton ("Common Internet File System (CIFS), Version: CIFS-Spec 0.9") in view of Miloushev (US 2002/0120763).

Claims 4, 6, 11, 12, 15, 19, 21, 26, 27, 29, 35, 37, 42-45, and 53 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Norton in view of Bourne (US 2003/0120875).

Claims 63 and 65 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Norton and Miloushev as applied supra, further in view of Bourne.

OVERVIEW OF CLAIMED INVENTION

The presently claimed invention provides for a locking system and method that implements publish consistency, and takes advantage of the SAN architecture. This is provided utilizing two whole file locks: a producer lock P and a consumer lock C. Any client can obtain a consumer lock at any time, but only a single writer is able to hold a producer lock.

Clients holding a consumer lock can read data and cache data for read. Clients holding a producer lock can write data, allocate and cache data for writing, with all writes being performed out-of-place. Because clients hold P and C locks concurrently, the protocol must ensure that the P lock does not overwrite any of the old version of the file. Out-of-place writing permits the clients holding the P lock to proceed with writing, while the system retains the old version.

File publication occurs when the P lock holder releases the lock. Upon release, the server notifies all clients of the new location of the file. It is important to note that clients read data directly from the SAN — servers do not ship the client's data as with AFS. Having told all clients about the new version of the file, the server reclaims the blocks that belong to the old version of the file that are not in the new version.

Clients that receive a publication invalidate the old version of the file and, when interested, read the new version from the SAN. The client need not invalidate the whole new file. Instead, it only needs to invalidate the portions of the file that have changed. Because changed blocks have been written out-of-place, the client knows that any portions of the file for which the location of the physical storage has not changed have not been written, and this cached copy is valid. There is also the opportunity for the server to compress the size of its publication message to clients by sending only the changed blocks.

In the Claims

An affidavit under 37 CFR 1.131 executed by each inventor is provided along with this amendment. The affidavit, along with the disclosure material provided as Exhibit A, establish

that the invention as claimed was conceived prior to March 26, 2001 and was coupled with due diligence from prior to March 26, 2001 to the filing of the patent application.

Based upon the facts established by the accompanying affidavit, applicants respectfully request the Examiner to withdraw all rejections where the cited prior art reference entitled, "Common Internet File System (CIFS), Version: CIFS-Spec 0.9" (also referred to as the Norton reference), is used. Specifically, rejections to claims 1-12, 14-32, 34-45, 58, and 61-67 are requested to be withdrawn as their rejection depends on the primary reference (Norton).

It should, however, be noted that the request to withdraw the rejections regarding claims 1-12, 14-32, 34-45, 58, and 61-67 does not indicate that the applicants acquiesce with the arguments put forth by the Examiner. For example, applicants contend that the primary reference (Norton) fails to provide many of the limitations of the claimed subject matter, some of which are listed below:

(1) Norton, either on its own or in combination with any of the cited references, fails to provide for the specific limitation of maintaining a consumer and a producer lock, wherein the consumer lock is maintained by one or more readers and the producer lock is maintained by a single writer.

(2) Norton, either on its own or in combination with any of the cited references, fails to provide for the specific limitation of writing data without publishing the same data (i.e., the updated file or blocks of data are published upon completion of update and upon release of the producer lock).

(3) Norton, either on its own or in combination with any of the cited references, fails to provide for the specific limitation of updating a cached copy of a file (associated with a reader) at a finer granularity by updating changed blocks of data.

(4) Norton, either on its own or in combination with any of the cited references, fails to provide for the specific limitation of allowing a writer with a producer lock to perform an out-of-place write to a storage unit that is physically separated from the file system server, wherein the storage unit is part of a storage area network

Hence, in view of the enclosed affidavit under 37 C.F.R. 1.131, the arguments presented during the interview, and the arguments presented above, applicants respectfully request the examiner to withdraw the rejections to claims 1-12, 14-32, 34-45, 58, and 61-67. It should be noted that the rejections with respect to claims 13, 33, 46-57, 59-60, and 68 are considered moot in view of their cancellation.

SUMMARY

As has been detailed above, none of the references, cited or applied, provide for the specific claimed details of applicants' presently claimed invention, nor renders them obvious. It is believed that this case is in condition for allowance and reconsideration thereof and early issuance is respectfully requested.

This Amendment is being filed with an extension of time for one month.

If it is felt that an interview would expedite prosecution of this application, please do not hesitate to contact applicants' representative at the below number.

Respectfully submitted,

Ramraj Soundararajan

Ramraj Soundararajan
Registration No. 53,832

1725 Duke Street
Suite 650
Alexandria, Virginia 22314
(703) 838-7683
February 27, 2004

APPENDIX A

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of: Burns et al.

Serial No.: 09/849,307 Group Art Unit: 2189

Filed: May 7, 2001 Examiner: Clifford H. Knoll

Title: *A Producer/Consumer Locking System for Efficient Replication of File Data*

AFFIDAVIT UNDER 37 CFR 1.131

MS Non-Fee Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

As a below inventor, I hereby declare that:

1) I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the patent application filed May 7, 2001 for A Producer/Consumer Locking System for Efficient Replication of File Data and inventor of the subject matter described and claimed therein.

2) Prior to March 26, 2001, I conceived the invention as described and claimed in the subject application and as amended by the amendment accompanying this affidavit in the United States as evidence by DISCLOSURE MATERIAL, attached hereto as Exhibit A.

3) From prior to March 26, 2001, until the filing of the patent application on May 7, 2001, I exercised due diligence toward reducing the invention to practice, as evidenced by DISCLOSURE MATERIAL, attached hereto as Exhibit A. The disclosure was evaluated and processed as part of IBM's standard patent processing procedures and culminated in the filing of the patent application on May 7, 2001.

4) The photocopies of DISCLOSURE MATERIAL attached to this affidavit as Exhibit A are true copies of the original pages showing conception of the invention prior to March 26, 2001 coupled with due diligence from prior to March 26, 2001 to the filing of the patent application.

Date February 16, 2004 _____ L.S.
Randal Chilton Burns

State of Maryland _____

County of Baltimore _____

Sworn to and subscribed before me this 16th day of February, 2004.

_____
Notary Public Jean Kajder Pugh

Date _____, 2004 _____ L.S.
Robert Michael Rees

State of _____

County of _____

Sworn to and subscribed before me this _____ day of _____, 2004.

_____
Notary Public

Date _____, 2004 _____ L.S.
Atul Goel

State of _____

County of _____

Sworn to and subscribed before me this _____ day of _____, 2004.

_____
Notary Public

Date _____, 2004 _____ L.S.
Wayne Curtis Hineman

State of _____

County of _____

Sworn to and subscribed before me this _____ day of _____, 2004.

_____
Notary Public

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of: Burns et al.

Serial No.: 09/849,307 Group Art Unit: 2189

Filed: May 7, 2001 Examiner: Clifford H. Knoll

Title: *A Producer/Consumer Locking System for Efficient Replication of File Data*

AFFIDAVIT UNDER 37 CFR 1.131

MS Non-Fee Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

As a below inventor, I hereby declare that:

1) I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the patent application filed May 7, 2001 for A Producer/Consumer Locking System for Efficient Replication of File Data and inventor of the subject matter described and claimed therein.

2) Prior to March 26, 2001, I conceived the invention as described and claimed in the subject application and as amended by the amendment accompanying this affidavit in the United States as evidence by DISCLOSURE MATERIAL, attached hereto as Exhibit A.

3) From prior to March 26, 2001, until the filing of the patent application on May 7, 2001, I exercised due diligence toward reducing the invention to practice, as evidenced by DISCLOSURE MATERIAL, attached hereto as Exhibit A. The disclosure was evaluated and processed as part of IBM's standard patent processing procedures and culminated in the filing of the patent application on May 7, 2001.

4) The photocopies of DISCLOSURE MATERIAL attached to this affidavit as Exhibit A are true copies of the original pages showing conception of the invention prior to March 26, 2001 coupled with due diligence from prior to March 26, 2001 to the filing of the patent application.

Date _____, 2004 _____ L.S.
Randal Chilton Burns

State of _____

County of _____

Sworn to and subscribed before me this _____ day of _____, 2004

_____
Notary Public

Date 2/23, 2004 _____ L.S.
Robert Michael Rees

State of Calfornda

County of Santa clara

Sworn to and subscribed before me this 23rd day of February, 2004.

Errol O. Smedley
Notary Public

ERROL O. SMEDLEY
Comm. # 1304907
NOTARY PUBLIC - CALIFORNIA
Santa Clara County
My Comm. Expires May 18, 2005

Date _____, 2004 _____ L.S.
Atul Goel

State of _____

County of _____

Sworn to and subscribed before me this _____ day of _____, 2004.

_____
Notary Public

Date 2/23, 2004 Wayne C. Hineman L.S.
Wayne Curtis Hineman

State of Calfornda

County of Santa clara

Sworn to and subscribed before me this 23rd day of February, 2004.

Errol O. Smedley
Notary Public

ERROL O. SMEDLEY
Comm. # 1304907
NOTARY PUBLIC - CALIFORNIA
Santa Clara County
My Comm. Expires May 18, 2005

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of: Burns et al.

Serial No.: 09/849,307 Group Art Unit: 2189

Filed: May 7, 2001 Examiner: Clifford H. Knoll

Title: *A Producer/Consumer Locking System for Efficient Replication of File Data*

AFFIDAVIT UNDER 37 CFR 1.131

MS Non-Fee Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

As a below inventor, I hereby declare that:

1) I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the patent application filed May 7, 2001 for A Producer/Consumer Locking System for Efficient Replication of File Data and inventor of the subject matter described and claimed therein.

2) Prior to March 26, 2001, I conceived the invention as described and claimed in the subject application and as amended by the amendment accompanying this affidavit in the United States as evidence by DISCLOSURE MATERIAL, attached hereto as Exhibit A.

3) From prior to March 26, 2001, until the filing of the patent application on May 7, 2001, I exercised due diligence toward reducing the invention to practice, as evidenced by DISCLOSURE MATERIAL, attached hereto as Exhibit A. The disclosure was evaluated and processed as part of IBM's standard patent processing procedures and culminated in the filing of the patent application on May 7, 2001.

4) The photocopies of DISCLOSURE MATERIAL attached to this affidavit as Exhibit A are true copies of the original pages showing conception of the invention prior to March 26, 2001 coupled with due diligence from prior to March 26, 2001 to the filing of the patent application.

Date_____, 2004 _____L.S.
Randal Chilton Burns

State of _____

County of _____

Sworn to and subscribed before me this _____ day of _____, 2004.

_____
Notary Public

Date_____, 2004 _____L.S.
Robert Michael Rees

State of _____

County of _____

Sworn to and subscribed before me this _____ day of _____, 2004.

_____
Notary Public

Date 10th FEBRUARY, 2004 Atul Goel L.S.
Atul Goel

State of CALIFORNIA

County of SANTA CLARA

Sworn to and subscribed before me this 10th day of Feb, 2004.

Pavneet Singh.
Notary Public

PAVNEET SINGH
Commission # 1318493
Notary Public - California
Santa Clara County
My Comm. Expires Aug 20, 2005

Date_____, 2004 _____L.S.
Wayne Curtis Hineman

State of _____

County of _____

Sworn to and subscribed before me this _____ day of _____, 2004.

_____
Notary Public

Disclosure ARC8-1999-0347

Created By: Randal Burns **Created On:** 11/09/99 03:41:12 PM
Last Modified By: Randal Burns **Last Modified On:** 12/17/99 02:52:51 PM

*** IBM Confidential ***

Required fields are marked with the asterisk (*) and must be filled in to complete the form .

Summary

Status	Under Evaluation
Processing Location	ARC
Functional Area	DPB - Computer Science - (A.K. Chandra)
Attorney/Patent Professional	Marc D McSwain/Almaden/IBM
IDT Team	Marc D McSwain/Almaden/IBM; Suzanne Pautlitz/Almaden/IBM
Submitted Date	12/02/99 01:30:07 PM
Owning Division	RES Add/Change
PVT Score	76
Incentive Program	(INC7) Interoperability in Heterogeneous Environments
Lab	
Technology Code	

Inventors with Lotus Notes IDs

Inventors: Randal Burns/Almaden/IBM, Bob Rees/Almaden/IBM, Atul Goel/India/IBM, Bruce Baumgart, Wayne Hineman/Almaden/IBM

Inventor Name > denotes primary contact	Inventor Serial	Div/Dept	Manager Serial	Manager Name
> Burns, Randal C.	849260	22/K56B	651497	Pease, David A.
Rees, Bob	690931	22/K56B	651497	Pease, David A.
Goel, Atul	911819	00/0053	918580	Raghavan, M.
Bruce Baumgart	N/A	N/A	N/A	N/A
Hineman, Wayne C.	106442	22/K56B	651497	Pease, David A.

Inventors without Lotus Notes IDs

IDT Selection

IDT Team:	Attorney/Patent Professional:
Marc D McSwain/Almaden/IBM Suzanne Pautlitz/Almaden/IBM	Marc D McSwain/Almaden/IBM

Response Due to IP&L : 01/16/2000

Main Idea

***Title of disclosure (in English)**

A Producer/Consumer Locking System for Efficient Replication of File Data

***Idea of disclosure**

1. Describe your invention, stating the problem solved (if appropriate), and indicating the advantages

of using the invention.
For reasons of manageability and ease of implementation, and despite the distributed file system's performance shortcomings, enterprise customers often choose to use file systems to replicate data among many web servers. Distributed file systems offer the web developer a well known interface that appears identical to a local file systen The interface shields the complexity of data distribution and consistency issues from the application. Web applica are written for a single server, and then deployed with no modifications on many computers. The deployment lev the enterprise's existing distributed file system, and no new distributed systems need to be installed or administer

The limitation of using a file system for wide area replication of web data is performance. Generally, file systems implement data locks that provide strong consistency between readers and writers. For replicating data from one writer to multiple readers, strong consistency produces lock contention -- a flurry of network messages to obtain and revoke locks. Contention loads the network and results in slow application progress.
In this invention, we present a new set of locks that drastically improve the efficiency of using a file system for w serving and show how a distributed file system can take advantage of these locks.

Dynamic web data has become increasingly prevalent in large scale web applications and enterprise web servers
Examples include IBM's Olympics web site, where race results and other data are dynamically added to a series of
web pages, and financial services web sites, where changing market values constantly modify the Internet content
being served. Many web servers must be able to serve this dynamic data to make the applications scale to
enterprise-wide and international data consumers. Using a file system for replication data allows web applications to
easily scale to many computers and our invention makes this system operate efficiently.

2. How does the invention solve the problem or achieve an advantage,(a description of "the invention", including figures inline as appropriate)?
Attached Acrobat PDF or postscipt file describes the invention in detail and compares it to similar


technologies. The invention is specified in Section 6. datalocks r datalocks n

3. If the same advantage or problem has been identified by others (inside/outside IBM), how have those others solved it and does your solution differ and why is it better?
This is addressed in the attached document.

4. If the invention is implemented in a product or prototype, include technical details, purpose, disclosure details to others and the date of that implementation.
The invention will be implemented in the Storage Tank distributed file system. This work is scheduled to be performed in 3Q2000. The details of the Storage Tank project can be seen at our internal web site (http://steer.almaden.ibm.com/cs/stortank/) .

***Critical Questions (Questions 1 - 7 must be answered)**

***Question 1**
On what date was the invention workable? 09/11/99 **Please format the date as MM/DD/YYYY** (Workable means i.e. when you know that your design will solve the problem)

***Question 2**	○ Yes ○ No
Is there any planned or actual publication or disclosure of your invention to anyone outside IBM?	

If yes, Enter the name of each publication or patent and the date published below.
Publication/Patent:
Date Published or Issued:

Are you aware of any publications, products or patents that relate to this invention? ● Yes ○ No

If yes, Enter the name of each publication or patent and the date published below.

```
@InProceedings{afscache,
   title = {Synchronization and Caching Issues in the {Andrew} File System},
   author = {M. ~L. Kazar},
   booktitle = {Proceedings of the {USENIX} Winter Technical Conference},
   month = feb,
   year = 1988
}

@inproceedings{dfs,
  author = {M. ~L. Kazar and B. ~W. Leverett and O. ~T. Anderson and
    V. Apostolides and B. ~A. Bottos and S. Chutani and
    C. ~F. Everhart and W. ~A. Mason and S. Tu and R. Zayas},
  title = {{DEcorum} file system architectural overview},
  booktitle = {Proceedings of the Summer {USENIX} Conference},
  month = jun,
  year = 1990
}
```

***Question 3**
Has the subject matter of the invention or a product incorporating the invention been sold, used internally in manufacturing, announced for sale, or included in a proposal? ○ Yes ● No

Is a sale, use in manufacturing, product announcement, or proposal planned? ○ Yes ● No

If Yes, identify the product if known and indicate the date or planned date of sale, announcements, or proposal and to whom the sale, announcement or proposal has been or will be made.
Product:
Version/Release:
Code Name:
Date:
To Whom:
If more than one, use cut and paste and append as necessary in the field provided.

***Question 4**
Was the subject matter of your invention or a product incorporating your invention used in public, e.g., outside IBM or in the presence of non-IBMers? ○ Yes ● No

If yes, give a date. **Please format the date as MM/DD/YYYY**

***Question 5**
Have you ever discussed your invention with others not employed at IBM? ○ Yes ● No

If yes, identify individuals and date discussed. Fill in the text area with the following information, the names of the individuals, the employer, date discussed, under CDA, and CDA #.

***Question 6**
Was the invention, in any way, started or developed under a government contract or project? ○ Yes ● No ○ Not sure

If Yes, enter the contract number

*Question 7 Was the invention made in the course of any alliance, joint development or other contract activities?	○ Yes ● No ○ Not Sure
If Yes, enter the following :Name of Alliance, Contractor or Joint Developer	
Contract ID number	
Relationship contact name	
Relationship contact E-mail	
Relationship contact phone	

Question 8 Have you submitted, or are you aware of, any related disclosure submission?	○ Yes ● No
If Yes, please provide the title and docket or disclosure number below:	

Question 9
What type of companies do you expect to compete with inventions of this type? *Check all that apply.*

- ☐ Manufacturers of enterprise servers
- ☐ Manufacturers of entry servers
- ☐ Manufacturers of workstations
- ☐ Manufacturers of PC's
- ☐ Non-computer manufacturers
- ☒ Developers of operating systems
- ☒ Developers of networking software
- ☒ Developers of application software
- ☒ Integrated solution providers
- ☐ Service providers
- ☐ Other (Please specify below)

Patent Value Tool (Optional - this may be used by the inventor and attorney to assist with the evaluati

(The Patent Value tool can be used by you or the evaluation team to determine the potential licensing value of your invention.)

These are the answers which were entered into the **Patent Value Tool.**

Market

What is the anticipated annual market size (in dollars) that will be captured by your invention?
Greater than $5B
Reason(s) for above Answer Market research indicates that the global market for enterprise storage will be $8 mil by 2002. This invention capture both enterprise storage and web serving, which is an even larger market.

CLAIMS

Question 1 - How new is the technical field?
Emerging
Reason(s) for above Answer While the marketplace is abuzz about SANs, the hardware is not mature enough to build the type of networks to which this invention applies. This type of maturity

is about 2 years out.
Question 2 - How central is the invention to the product(s) which might be expected to contain the invention?
Essential
Reason(s) for above Answer This is the key technology that makes the Storage Tank distributed file system a viable solution for wide area replication in Web Servers. Without the invention, Storage Tank is just a file system.
Question 3 - What is the scope of the claim?
Fundamental
Reason(s) for above Answer The claim is to make file systems a viable solution for wide area replication of data for Web Serving. Again, without the invention, this product could not differentiate itself from other distributed file systems.

PORTFOLIO NEED View PPM Needs List
What are the portfolio needs in the area of your invention?
Listed in PPM Needs
Reason(s) for above Answer Covered by areas 300 and 600.

EXPLOITATION & ENFORCEMENT
Question 1 - How easily can the use of the invention by a competitor be detected?
Easily
Reason(s) for above Answer The invention eliminates messages between a web server and a storage server. This should be detected by observing network messages.
Question 2 - How easily can the use of the invention be avoided by a competitor?
Unavoidable
Reason(s) for above Answer The performance improvement provided by the invention are not available without use of the invention. This is a side effect of the broadness of the claims.

BUSINESS VALUE
Question 1 - What percentage of the companies producing products in the field of this invention might use this invention?
Broadly cloned
Reason(s) for above Answer Any company producing storage servers for the enterprise and web solutions would benefit from this technology.
Question 2 - What is the value of this patent to current or anticipated Alliance Activity between IBM and other companies?
High value
Reason(s) for above Answer Enterprise storage and web serving is both a competitive landscape with many players and of increasing importance to IBM's future plans.
Question 3 - What is the value of this patent to current or anticipated Technology Transfer Activity between IBM and other companies?
High value
Reason(s) for above Answer sames as question 2.
Question 4 - Does it result in prestige to IBM?
Industry wide
Reason(s) for above Answer This technology will results in a special function to improve the performance, scalability and ease of use for replicated web servers. This can be easily captured and individually distinguished in a product. For the same reason, the invention can easily be marketed. This invention should provide a competitive advantage to IBM in the server business.

Post Disclosure Text & Drawings
Enter any additional information relating to this disclosure below:

(Form Revised 12/17/97)

Data Locking in the Storage Tank Distributed System

Abstract

The file system interface is commonly used to access distributed storage because it is simple and well understood by application developers. However, the applications that use this interface have different requirements and performance characteristics. For this reason, the Storage Tank distributed system provides multiple performance and consistency qualities of service. Storage Tank does this by implementing multiple distributed locking protocols for data consistency and cache coherency, each with unique semantics.

1 Introduction

Distributed file systems have become the principal method for sharing data in distributed applications. Programmers understand local file system semantics well and use them to gain access to shared data easily. For exactly the same reason that distributed file systems are easy to use, they are difficult to implement. The distributed file system takes responsibility for providing synchronized access and consistent views of shared data, shielding the application and programmer from these tasks, by moving the complexity into the file system.

The file system responsibilities include data consistency and cache coherency. In a file system, the data consistency guarantee describes the interaction between distributed clients that concurrently read and write the same data. Most often, file systems implement sequential consistency [17] – "a multi-processor is *sequentially consistent* if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor occur in this sequence in the order specified by its program." Sequential consistency is the strongest consistency guarantee. However, some applications benefit from weakening the consistency guarantee in order to achieve better performance.

Cache coherency is the task of making sure that the data cached by a client does not violate the consistency guarantee and generally requires a locking protocol. For cache coherency protocols, the semantics of the locks used to implement the protocol define how data is managed in the system. However, in general, cache coherency locking is designed the other way around. Developers of a distributed data system choose the data model appropriate to their expected workload, and implement a set of locks that conform to that model.

For many distributed applications, the consistency guarantee and caching provided by the file system is more than adequate. These applications write to the file system interface and ignore all of the complexity of managing distributed data.

Other distributed applications are not compatible with the file system consistency and coherency model, but would still like to take advantage of other file system services. A file system provides many things beyond consistency and caching, including: a name space, through which objects can be accessed; data management functions, such as backup and restore and hierarchical storage management; data reliability and availability, striping [4, 13] and RAID [7]; and distributed logical volume support, mapping logical addresses to physical devices.

In Storage Tank, we wish to make our distributed storage management useful to as many applications as possible. Our approach is to provide multiple quality of service options through locking. Storage Tank offers a locking system designed for sequential consistency with write-back caching, typical of distributed file systems. We also provide a locking system for sequential consistency with no caching for applications that manage their own caches. Finally, we include a locking system that implements a weaker consistency model with write-back caching, designed for efficient replication and distribution of data. Locks for replication are suitable for serving dynamic data on the Internet and other highly-concurrent applications.

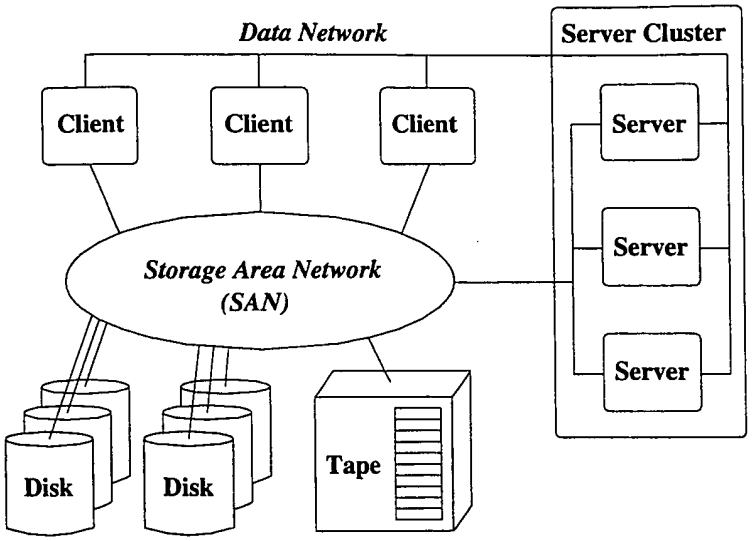


Figure 1: Schematic of the Storage Tank Client/Server Distributed File System on a Storage Area Network (SAN).

2 Storage Tank

A brief digression into the file system architecture in which we implement locking helps to explain the motivation for and advantages of multiple protocols.

In the Storage Tank project at IBM research, we are building a distributed file system on a storage area network (SAN) (Figure 1). A SAN is a high speed network designed to allow multiple computers to have shared access to many disk drives. Currently, storage area networks are being constructed on Fibre Channel (FC) networks [3] and IBM's Serial Storage Architecture. In the future, we expect network-attached storage devices to be available for general purpose data networks, so that SANs can be constructed for high-speed networks such as Gigabit Ethernet [8]. A distributed file system built on a SAN removes the server bottleneck for I/O requests by giving clients a direct data path to disks.

When building a distributed file system on a SAN, file system clients can access data directly over the storage area network. Most traditional client/server file systems [21, 18, 16, 6] store data on the server's private disks. Clients function ship all data requests to a server that performs I/O on their behalf. Unlike traditional network file systems, Storage Tank clients perform I/O directly to shared storage devices. This direct data access model is similar to the file system for Network Attached Secure Disks [9], using shared disks on an IP network, and the Global File System [20], for SAN-attached storage devices. In Storage Tank, clients access storage devices directly over the SAN to obtain data. Clients communicate with Storage Tank servers over a separate general purpose network to obtain file metadata. In addition to serving file system metadata, the servers manage cache coherency protocols, authentication, and the allocation of file data (managing data placement and free space).

Unlike most file systems, Storage Tank's metadata and data are stored separately. Metadata, including the location of the blocks of each file on shared storage, are kept on high-performance storage at the server. The shared disks contain only the blocks of data for the files and no metadata. In this way, the shared devices on the SAN can be optimized for data traffic (block transfer of data), and the server private storage can be optimized for metadata workload (frequent small reads and writes).

The SAN environment simplifies the distributed file system server by removing its data tasks and radically changes the server's performance characteristics. Previously, servers performance for a distributed

file system was measured by data rate (Megabytes/second). Performance was occasionally limited by network bandwidth, but more often limited by a server's ability to read data from storage and write it to the network. In the SAN environment, without any storage or network I/O, a server's performance is more properly measured in transactions per second, analogous to a database server. Without data to read and write, the Storage Tank file server performs many more transactions than a traditional file server with equal processing power. By using computational resources more efficiently, Storage Tank should present a higher data rate to the file system client and allow a distributed file system to fully utilize its network infrastructure with less investment in server hardware.

3 Background

This work describes only the semantics of sets of locks and does not discuss lock management, *i.e.* the granting and revoking of locks in a distributed system. We also assume that the reader is familiar with locking data structures, the concepts of lock upgrade and downgrade, deadlock and data caching. Refer to *Transaction Processing* [11] for a treatment of locking data structures and lock semantics and to *Parallel Computer Architecture* [5] for a discussion of caching and coherency protocols.

4 File System Locking

Storage Tank provides a set of locks designed for the traditional file system workload. These locks implement sequential consistency and allow clients to cache data for both reading and writing. This consistency and caching model is identical to that used by major distributed file systems like DFS [16] and Calypso [6].

In the traditional file system workload, data are shared infrequently [2, 15], which means that any given file should have a strong affinity to the client that uses it. This would imply that clients should lock and unlock data for whole files at a time. While this statement holds true for the majority of files, when write sharing occurs, locking data at a finer granularity than the file level reduces lock contention. In Storage Tank, as with other file systems [14], we use hierarchical data locks to support sub-file or segment locking when write sharing occurs and allow whole files to be locked with a small amount of lock state when data are not shared for writing. This system is identical in concept to granular locking in database tables [11].

In Storage Tank, each file is divided into a collection of segments. Segments are logical and partition the address space of a file into pieces of fixed width. Segments are the locking granule. For each segment, we define two preemptible cache locks. The shared segment lock S_S allows lock holders to read data and cache data for read. The exclusive segment lock S_X permits the lock holder to write data and cache data for write. Because segments are logical, not physical, segments that do not contain any data can be locked. For example, it is legal to lock a segment that is past the end of the file. That segment contains no data and has no physical storage allocated for it when it is locked.

The exclusive segment lock can be held by only one client at a time, and no shared locks on the same segment may be concurrently held. A shared segment lock can be held by multiple clients at the same time. The compatibility table for these locks appears in Table 1.

Certain attributes of a file cannot be captured by a single segment lock and require ownership of the whole file. For example, a process that truncates a file to zero length must hold a lock that grants exclusive access to every segment of the file. It is not adequate to perform whole file operations under multiple segment locks because operations, like truncation, must occur atomically. For this purpose, Storage Tank also provides a set of locks that apply to the whole file. Whole file locks prevent the deadlock and starvation problems that arise when trying to lock every segment of a file individually. We provide an exclusive lock L_X and a shared lock L_S on the whole file. In addition to whole file locks, we provide shared (L_{IS}) and exclusive (L_{IX}) intention locks [11]. We also provide a shared whole file lock combined with an intention

Granted Mode			
Requested Mode	None	S_S	S_X
S_S	+	+	−
S_X	+	−	−

Table 1: Compatibility table for the shared and exclusive segment locks.

Granted Mode						
Requested Mode	None	L_{IS}	L_{IX}	L_S	L_{SIX}	L_X
L_{IS}	+	+	+	+	+	−
L_{IX}	+	+	+	−	−	−
L_S	+	+	−	+	−	−
L_{SIX}	+	+	−	−	−	−
L_X	+	−	−	−	−	−

Table 2: Compatibility table for file locks.

to lock individual segments exclusively (L_{SIX}) whose privileges are merely the union of L_S and L_{IX}. Intention locks are obtained by clients that "intend" to lock segments individually. A client that wishes to lock individual segments in a shared mode must first obtain a shared intention lock. Similarly, to lock segments in an exclusive mode, an exclusive intention lock must be held. The compatibility of these locks is described in Table 2, where the +'s indicate that the requested lock mode can be granted in conjunction with the currently held lock, and the −'s indicate that the request is in conflict with the current lock state.

Holders of the exclusive lock L_X can modify the length of the file, can read and write any segment, and can cache any segment for read or write. Holders of the shared lock are allowed to read any portion of the file, and are guaranteed that the length of the file does not change.

Segment and whole file locks have a hierarchical relationship that dictates their management. Only when a client holds an L_{IX}, L_{IS}, or L_{SIX} file lock can it obtain a shared segment lock. Similarly, only in the context of an L_{IX} or L_{SIX} lock can an exclusive segment lock be obtained. So, as a side effect, whenever any intention lock is revoked, all of its subordinate segment locks must be relinquished.

The intent of this locking scheme is to operate efficiently for both regular file system workload, with little sharing, and the less frequent occurrence of concurrent write sharing. When clients are writing data that changes the length of the file, they hold whole file locks. For clients that read data, they generally hold locks for the segments that they are reading. The segment lock granule is the same as cache management unit, and segment locking captures locality in the cache. All of these locks allows clients to cache data for the actions the lock allows. Readers cache data at a segment of file granularity for read. Writers cache data for writing, *i.e.* operate a write-back cache. Aggressive client-side caching is a widely held tenet in distributed file system design for file system workload. We implement a set of data locks that allows for client side caching with a sequential consistency guarantee on the data.

5 Locking for Databases and Parallel Applications

To optimize performance, availability and recovery, databases and other distributed applications manage their own data caches. Applications have knowledge of the structure of the data on which they operate and the semantics with which the data are changed. Consequently, applications can cache data at a finer granularity than a file system can, and therefore achieve more parallelism. They also can customize their

Mode	File System Locking	Database and Parallel Application Locking
S_S	R	RW
S_X	RWA	RWA
L_{IS}	S_S	S_S
L_{IX}	S_X	S_X
L_S	R	RW
L_{SIX}	RS_X	RWS_SX
L_X	RW	RWA

Table 3: A comparison of the file operations allowed under locks between file system locking and no-caching locking.

distributed application's caching properties, *e.g.* implement a cooperative cache [1, 12] or a different consistency guarantee [10].

Implementing these applications on file systems that cache data results in data being cached twice. Also, file systems that cache written data prevent application caches from realizing as much parallelism as possible and reduce performance. The Storage Tank distributed file system provides applications that cache data access to distributed and shared storage without caching.

Many databases and parallel I/O applications [19] use either a physical device or logical volume interface to avoid caching and obtain good parallel data performance. While device or volume access solves performance problems, consumers often prefer to build databases in file systems for manageability reasons. The advantages of a file system include name space, integrated backup, and extensibility. File systems give a hierarchically organized name to database objects, which is easier to understand than volume or device names. Data stored in file systems can be copied, moved, or backed up with file system tools, and databases offload this function to the file system. Finally, files make databases easy to extend and do not require an administrator for this task.

We use an example to illustrate the manner in which caching at the file system limits parallelism. Assume an application that caches its own data at a fine granularity, a cache unit of 4k bytes, constructed on a file system that caches data in larger segments. Now, if two clients concurrently write separate application cache units, they expect the writes to occur in parallel. However, because both writes lie within the same file system cache segment, concurrent writes are executed serially. That is, the first client to write to the file system obtains a write lock for that segment, and writes the data to the file system's cache. When the second client attempts to write, the write lock of the first client is revoked. The first client commits its changes to disk. The second client must then read the segment from disk before applying its write to the segment. Because the file system segment is larger than the application cache segment, writes often need to be serialized.

To alleviate doubling caching and increase parallelism, we offer a set of strong consistency locks without caching – *no-caching locks*. In this mode, we say that the file system "stays out of the way" of parallel applications and databases that manage caches within their applications. All I/O operations submitted to the file system are executed synchronously with stable storage.

To achieve strong consistency locks without caching, we use the same locking modes as we did for segment and whole files with caching (Section 4), but the client actions under these locks differ. We say that we use the "same locks", because the locks have the same names, compatibility tables, and semantics. However, because there is no caching, the clients can take different actions under the same locks.

The differences between locking for databases and parallel applications and locking for file systems are summarized in Table 3. In this table, allocation refers to a client changing the storage that backs a portion of file address space. In regular file system workload, allocation need not be differentiated from write; the

write privilege always implies an allocation privilege. However, for database and parallel applications, the write privilege is granted on a shared lock and must be differentiated from allocation. Write, in this case, means an *in-place* write, where data are written back to the same physical location from which they were, or could have been, read. This is to be differentiated from *out-of-place* write, where portions of a file are written to physical storage locations that differ from the locations from which they were read. To be precise, in our terms, allocation means placing physical storage behind logical file address space, and writing means writing data to an existing allocation. An out-of-place write should properly be considered an allocation followed by a write.

The changes in the lock semantics are systematic – all shared locks can now write data, but cannot allocate. Because there is no caching at the file system, all I/O operations go directly to a single non-volatile copy of the data, and multiple concurrent writers do not violate sequential consistency. The only guarantee multiple concurrent writers need is that the data does not change location, hence allocation requires an exclusive lock.

No-caching locks allow applications to manage caches directly. Databases and parallel applications using these locks realize the parallelism and efficiency expected when performing I/O to raw devices or logical volumes. These applications also receive the manageability advantages of a file system.

6 Locking for Wide Area Replication

For reasons of manageability and ease of implementation, and despite the distributed file system's performance shortcomings, enterprise customers often choose to use file systems to replicate data among many web servers. Distributed file systems offer the web developer a well known interface that appears identical to a local file system. The interface shields the complexity of data distribution and consistency issues from the application. Web applications are written for a single server, and then deployed with no modifications on many computers. The deployment leverages the enterprise's existing distributed file system, and no new distributed systems need to be installed or administered.

The limitation of using a file system for wide area replication of web data is performance. Generally, file systems implement data locks that provide strong consistency between readers and writers. For replicating data from one writer to multiple readers, strong consistency produces lock contention – a flurry of network messages to obtain and revoke locks. Contention loads the network and results in slow application progress. We present an example application to illustrate lock contention, propose a new set of locks that address this problem, and show how Storage Tank's architecture takes unique advantage of these locks.

6.1 Strong Consistency

We use posting Olympics race results as an example application to show how strong consistency locking works for replicating web servers. A possible configuration of the distributed system (Figure 2) has hundreds of web servers integrated with a DBMS. Race results and other dynamic data are inserted into the database through an interface like SQL or ODBC. The insertion updates database tables, which sets off a database trigger. The trigger causes the databases to author a new web page. The file system is responsible for distributing the new version of the web page to the remainder of the web servers in a consistent fashion. Note that I/O are performed out of file systems caches, which the locking protocol keeps consistent. Web servers take HTTP requests, which result in data reads from the file system.

Poor performance occurs for a strong consistency locking protocol when updates occur to active files. For this example, we assume that the file that is being updated is being read concurrently by multiple web servers before, during, and after new results are being posted and written. For race results during the Olympics, a good example would be a page that contains a summary of how many gold, silver and bronze

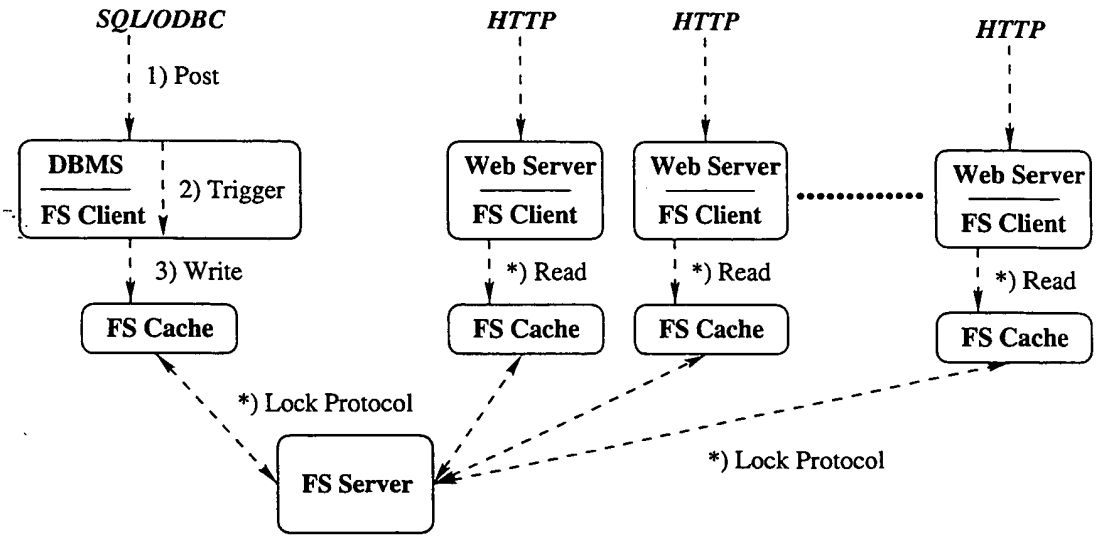


Figure 2: Example configuration of using a file system to provide replication for large scale web serving.

medals have been won by each country. This page has wide interest and changes often. We use the strong consistency locks of Section 4 for our example. The system's initial configuration has the file system clients at all web servers (denoted by Web Server*) holding a shared lock for the file in question. Figure 3 displays the locking messages required to update a web page in a timing diagram

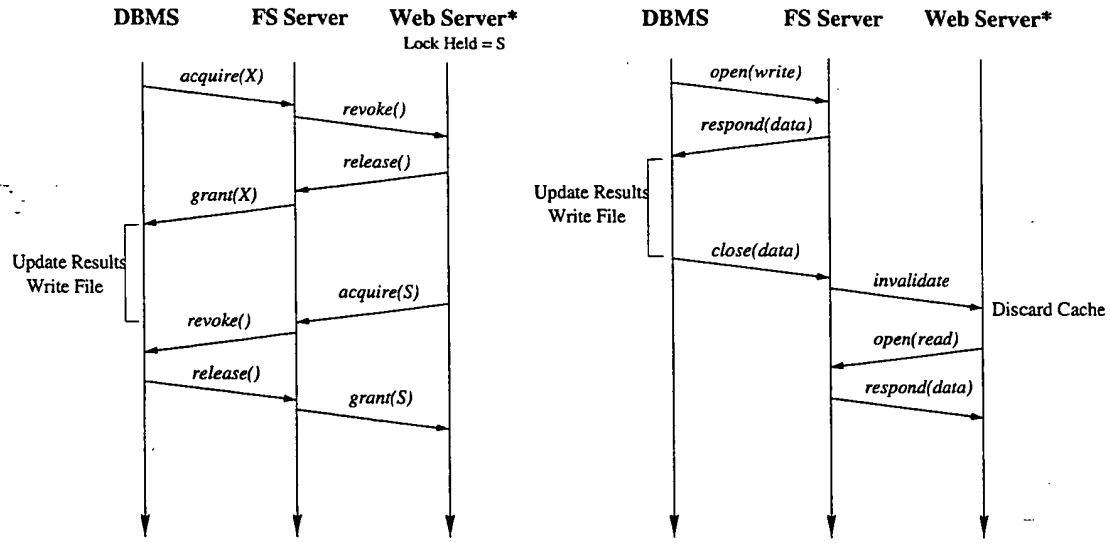
The best possible performance of strong consistency locking occurs when the file system client at the database is not interrupted while it updates the web page. In this case, the writing client requests an exclusive lock on the file, which revokes all concurrently held read locks. When the writer completes, the client at each web server must request a shared lock on the data to reobtain the privilege to read and serve the web page. All messages to and from the web server occur for every web server. In the best case, four messages, revoke, release, acquire, and grant, go between each web server and the file system server. For large installations, multiple messages between web servers and the file server consume time and network bandwidth prohibitively.

The situation becomes much worse when the DBMS is interrupted in the middle of updating the file. Data locks are preemptible, so clients that request read locks on the file, revoke the DBMS's exclusive lock. The more highly replicated data are, the more web servers there are, and the more likely the DBMS will get interrupted. Contention for the lock can stall the update indefinitely. Furthermore, if the DBMS is interrupted in the middle of writing the new file, the update is incomplete, and the data that a reading client (Web Server) sees in not a parsable HTML document.

The example application presents a non-standard workload to the file system. The workload lacks the properties a file system expects, and therefore operates inefficiently. For example, the workload does not have spatial locality to clients. Instead, all clients are interested in all files.

Performance concerns aside, strong consistency is the wrong model for updating web pages. Reading clients cannot understand intermediate updates and byte-level changes. Instead, it would be desirable for web servers to continue serving the old version of the file (web page) while the DBMS updates the page. Then, when the DBMS finishes its update, the new page can invalidate the old page atomically. We call this a *publish* consistency model. Not only does publishing guarantee that web servers always see parsable HTML data, it also results in fewer messages.

Outside the intermediate update problem, strong consistency is still not useful for serving the HTTP protocol. Since the web client/server protocol (HTTP) does not implement consistency between the browser



(a) Strong consistency.

(b) AFS weak consistency

Figure 3: Locking protocols for distributing data to web servers through a file system.

cache and the server, a strong consistency guarantee at the server cannot be passed on to the ultimate consumers of the data at their browsers. Therefore, a weaker consistency model [10] looks identical to web users and can be more efficiently implemented.

6.2 AFS Consistency

For existing distributed systems, the Andrew file system (AFS) comes closest to publish consistency. AFS does not implement strong consistency, rather it chooses to synchronize file data between readers and writers when files opened for write are closed [15]. AFS cites statistics that argue that data are shared infrequently to justify this decision. The successor product to Andrew, the DEcorum file system [16], currently known as DFS, reversed this decision, because they found that distributed applications sometimes require sequential consistency for correctness. While strong consistency is now the de facto standard for file systems, AFS-style consistency remains useful for environments where concurrent action and low overhead dominate the need for correctness.

Figure 3 shows the protocol used in AFS to handle the dynamic web updates in our example. At the start of our timing diagram, all web servers hold the web page (file) in question open for read. In AFS, an open instance registers a client for *callbacks* – messages from the server invalidating their cached version. The DBMS opens the file for writing, writes the data to its cache, and closes the file. On close, the DBMS writes it cached copy of the file back to the server. The file system server notifies the web servers of the changes to the file by sending invalidation messages to all file system clients that have registered for a callback.

When compared to the protocol for strong consistency locking, AFS saves only one message between client and server. However, this belies the performance difference. In AFS, all protocol operations are asynchronous – file system clients never wait for lock revocations before proceeding. AFS eliminates lock contention and eliminates waiting for the locking protocol. The DBMS obtains a write instance from the server directly, and need not wait for a synchronous revocation call to all clients. Another significant advantage of AFS is that the old version of the web page is continuously available at the web servers while the file

Granted Mode			
Requested Mode	None	C	P
C	+	+	+
P	+	+	−

Table 4: Compatibility table for producer and consumer locks.

is being updated at the DBMS.

The disadvantage of AFS is that it does not correctly implement publish consistency. The actual behavior is that the AFS clients write dirty data back to the server when closing a file, and AFS servers send callback invalidation messages whenever clients write data. In most cases, these policies result in publish consistency. However, if a writing client writes back some portion of its cache without closing the file, a callback is sent to all registered clients, and reading clients can see partial updates. This most often occurs when a writing client, in our example the DBMS, operates under a heavy load or on large files. In these cases, the cache manager writes back dirty blocks to the server to reclaim space.

6.3 Publish Consistency

For Storage Tank, we offer a locking option for replication that captures the performance efficiency of AFS, implements publish consistency exactly, and takes advantage of the SAN architecture. We capture all of this in two whole file locks: a producer lock P and a consumer lock C. Any client can obtain a consumer lock at any time and a producer lock is held by a single writer (Table 4). Clients holding a consumer lock can read data and cache data for read. Clients holding a producer lock can write data, allocate and cache data for writing, with the caveat that all writes are performed out-of-place. Because clients hold P and C locks concurrently, the protocol must ensure that the P lock does not overwrite any of the old version of the file. Out-of-place writing permits the client holding the P lock to proceed with writing, while the system retains the old version.

File publication occurs when the P lock holder releases his lock. Upon release, the server notifies all clients of the new location of the file. Recall (Section 2) that clients read data directly from the SAN – servers do not ship the clients data as with AFS. Having told all clients about the new version of the file, the server reclaims the blocks that belong to the old version of the file that are not in the new version.

Clients that receive a publication invalidate the old version of the file and, when interested, read the new version from the SAN. The client need not invalidate the whole new file. Instead, it only needs to invalidate the portions of the file that have changed. Because changed blocks have been written out-of-place, the client knows that any portions of the file for which the location of the physical storage has not changed have not been written, and its cached copy is valid. There is also the opportunity for the server to compress the size of its publication message to clients by sending only the changed blocks. Because location data are small and it makes server code more complex, we did not implement this feature.

Looking at out example using C and P locks (Figure 4), we see that the protocol uses fewer messages and all operations are asynchronous, like AFS. The DBMS requests and obtains a producer (P) lock from the server, which is granted immediately. While the DBMS holds the P locks, web servers continue to read and serve the current version of the file. When the DBMS completes its updates, it closes the file and releases the P lock. The writer must release the P lock on close to publish the file. The server sends location updates to all clients that hold consumer locks. The clients immediately invalidate the affected blocks in their cache. At some later time, when clients generate interest in the changed data, they read the data from the SAN.

The producer/consumer model has many of the same advantages of AFS and correctly implements

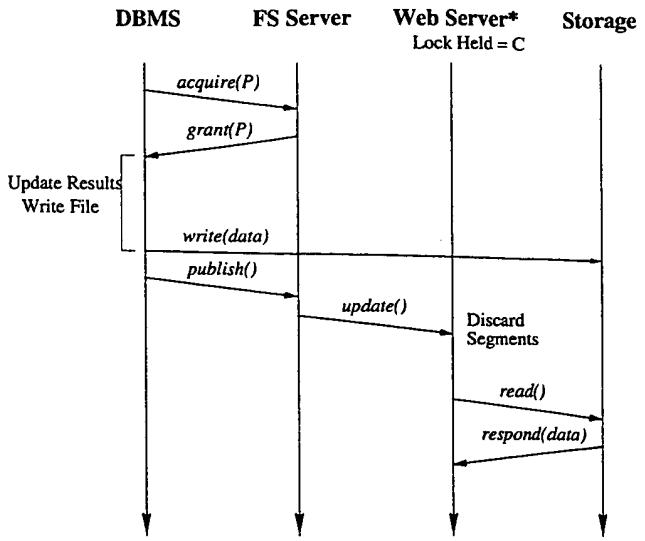


Figure 4: Producer/consumer publishing consistency protocol for distributing data to web servers.

publish consistency. Like AFS, all operations are asynchronous, read data is available while updates occur, and a client/server message is eliminated as compared to the strong consistency protocol. AFS overloads the DBMS writing data to the server as a cue to update the web server's data. With C and P locks, the web servers are not updated until the the P lock holder releases the lock, generally on closing the file. This means that the P lock holder can write data to the SAN and even update the location metadata at the server without publishing the data. C and P locks dissociate updating readers' cache contents from writing data, and, therefore, correctly implement publish consistency.

Producer/consumer locking also leverages the SAN environment to improve performance. First, network operations are distributed across two networks, an IP network for protocol messages and a SAN for data traffic. This mitigates the network bottleneck that arises when servers ship data over the same network on which they conduct the locking protocol. The two networks also allow clients to update data in their cache asynchronously and at a fine granularity. Unlike AFS, the whole file need not be replaced at once. Instead, because C lock updates invalidate cached data at a fine granularity, changes to data are read on a block by block basis from storage. This reduces the total amount of data involved in publishing the new file version and spreads the acquisition of that data over more time.

Storage Tank makes a hidden assumption in producer/consumer locking. Storage Tank requires the SAN storage to have a caching controller, *i.e.* storage devices cache data for read. The Web server's read data directly from storage, and they all read the same blocks. When caching data, the storage controller services reads at memory speed, rather than at disk speed. Without a cache, all reads would go to disk and any performance advantages of C and P locks would be lost in the slow I/O operations.

7 Selecting Lock Service

Because Storage Tank has multiple locking protocols, it must have a mechanism for selecting the protocol for a given file. Our general approach is that the locking protocol is selected through file metadata. When clients open a file, they receive metadata for that file that dictates which of the three locking protocols apply.

Assigning the locking protocol metadata when creating a file is the remaining problem. We address this through several techniques. Administrators set up default locking protocols for portions of the file system

name space, with strong consistency locking as the global default. Administrators also configure locking protocols based on policy – examples of selection criteria include owner, group, and file extension. Policy overrides name space selection. Finally, client applications can change the locking protocol after file creation through a file system ioctl call. Applications that require a quality of service can request it specifically by using a Storage Tank specific function call.

8 Conclusions

Many applications choose distributed file systems for ease of use and manageability. This results in file systems being used for many applications that were not originally considered and that do not exhibit traditional file system workload. To meet the needs of these applications, Storage Tank provides multiple qualities of service for consistency and data caching through its locking mechanism. In particular, Storage Tank implements different locking protocols tuned to the needs of the many different file system applications – file serving, parallel applications that manage their own cache, databases, Internet enterprise servers, and highly-concurrent replicating applications.

References

[1] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang. Serverless network file systems. *ACM Transactions on Computer Systems*, 14(1):41–79, February 1996.

[2] M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout. Measurements of a distributed file system. In *Proceedings of the 13th Annual Symposium on Operating Systems*, October 1991.

[3] A. F. Benner. *Fibre Channel: Gigabit Communication and I/O for Computer Networks*. McGraw-Hill Series on Computer Communications, 1996.

[4] L.-F. Cabrera and D. D. E. Long. Swift: Using distributed disk striping to provide high I/O data rates. *Computing Systems*, 4(4):405–436, 1991.

[5] D. E. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, Inc., San Francisco, California, USA., 1999.

[6] M. Devarakonda, D. Kish, and A. Mohindra. Recovery in the Calypso file system. *ACM Transactions on Computer Systems*, 14(3), August 1996.

[7] A. L. Drapeau, K. W. Shirriff, J. H. Hartman, E. L. Miller, S. Seshan, R. H. Katz, and D. A. Patterson. RAID-II: A high-bandwidth network file server. In *Proceedings of the 21st International Symposium on Computer Architecture*, 1994.

[8] H. Frazier and H. Johnson. Gigabit ethernet: From 100 to 1,000 Mbps. *IEEE Internet Computing*, 3(1), 1999.

[9] G. A. Gibson, D. F. Nagle, K. Amiri, F. W. Chang, H. Gobioff, E. Riedel, D. Rochberg, and J. Zelenka. Filesystems for network-attach secure disks. Technical Report CMU-CS-97-118, School of Computer Science, Carnegie Mellon University, July 1997.

[10] R. A. Golding. *Weak-consistency group communication and membership*. PhD thesis, University of California at Santa Cruz, 1992.

[11] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, Inc., San Mateo, California, USA., 1993.

[12] B. Gronvall, A. Westerlund, and S. Pink. The design of a multicast-based distributed file system. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, 1999.

[13] J. H. Hartman and J. K. Ousterhout. The Zebra-Striped network file system. In *Proceedings of the 14th ACM Symposim on Operating System Principles*, pages 29–43. ACM, 1993.

[14] R. Hasking and F. Schmuck. The tiger shark file system. In *Proceedings IEEE 1996 Spring COMPCON*, 1996.

[15] M. L. Kazar. Synchronization and caching issues in the Andrew file system. In *Proceedings of the USENIX Winter Technical Conference*, February 1988.

[16] M. L. Kazar, B. W. Leverett, O. T. Anderson, V. Apostolides, B. A. Bottos, S. Chutani, C. F. Everhart, W. A. Mason, S. Tu, and R. Zayas. DEcorum file system architectural overview. In *Proceedings of the Summer USENIX Conference*, June 1990.

[17] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, C-28(9):241–248, 1979.

[18] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. H. Rosenthal, and F. D. Smith. Andrew: A distributed personal computing environment. *Communications of the ACM*, 29(3):184–201, March 1986.

[19] MPI-2: Extensions to the message passing interface. Technical report, Message Passing Interface Forum, July 1997.

[20] K. W. Preslan, A. P. Barry, J. E. Brassow, G. M. Erickson, E. Nygaard, C. J. Sabol, S. R. Soltis, D. C. Teigland, and M. T. O'Keefe. A 64-bit, shared disk file system for Linux. In *Proceedings of the Sixteenth IEEE Mass Storage Systems Symposium*, 1999.

[21] D. Walsh, B. Lyon, G. Sager, J. Chang, D. Goldberh, S. Kleiman, T. Lyon, R. Sandberg, and P. Weiss. Overview of the Sun network file system. In *Proceedings of the 1985 Winter Usenix Technical Conference*, January 1985.